DiaSim: A Parameterized Simulator for Pervasive Computing Applications

Wilfried Jouve, Julien Bruneau, Charles Consel
INRIA / LaBRI / ENSEIRB, Talence, France
jouve@labri.fr, bruneau@enseirb.fr, consel@labri.fr

***Abstract*—Pervasive computing applications involve both software concerns, like any software system, and integration concerns, for the constituent networked devices of the pervasive computing environment. This situation is problematic for testing because it requires acquiring, testing and interfacing a variety of software and hardware entities. This process can rapidly become costly and time-consuming when the target environment involves many entities.**

In this demonstration, we present DiaSim, a simulator for pervasive computing applications. To cope with widely heterogeneous entities, DiaSim is parameterized with respect to a description of a target pervasive computing environment. This description is used to generate both a programming framework to develop the simulation logic and an emulation layer to execute applications. Furthermore, a simulation renderer is coupled to DiaSim to allow a simulated pervasive system to be visually monitored and debugged.

I. Introduction

Pervasive computing applications coordinate a variety of networked entities collecting context data from sensors and reacting by triggering actuators. To collect context data, sensors process stimuli that are observable changes of the environment (*e.g.,* fire and motion). Developing a pervasive computing application requires to address a number of issues such as entity heterogeneity, physical constraints, and types of stimuli present in the target environment. Also, such an application needs to implement strategies to manage a variety of scenarios *e.g.,* fire situations, intrusions, and crowd emergency-escape plans. Consequently, in addition to the challenges of developing any software system, a pervasive computing system needs to validate the application building blocks both individually and globally, to identify potential conflicts. In practice, the many parameters to take into account for the development of a pervasive computing application can considerably lengthen this process and a fully-equipped pervasive computing environment is still required to run and test an application. As a result, an iteration process is needed, involving the physical layout of the target environment and the application code. This process is cumbersome and hinders testing applications against a wide range of scenarios. To ease this process, we propose DiaSim, a parameterized simulator for pervasive computing applications.

II. Our Approach

DiaSim relies on the DiaGen approach [1], [2] where a high-level specification of a pervasive computing environment written in the DiaSpec language is passed to a compiler to produce a customized programming framework. This programming framework provides developers with high-level abstractions to discover services and to communicate with them. In the DiaSim approach, DiaGen automates the production of simulation environments, enables transparent simulation of applications and allows actual entities to be tested together with simulated ones. In the rest of this section, we describe the key features of our approach.

Parameterized simulator. Pervasive computing systems target a variety of application areas, including home automation, building surveillance and assisted living. A simulation tool for the pervasive computing domain is required to deal with different application areas, enabling new classes of entities and stimuli to be introduced easily. To adapt to various application areas, the DiaSim simulator is parameterized with respect to a high-level specification of a pervasive computing environment written in the DiaSpec language.

Transparent simulation. Our approach makes it possible for the same code to be simulated or executed in the actual environment. DiaSim emulate the execution of an application without requiring any change in the application code. As a result, when the testing phase is completed, the application code can be uploaded as is and its logic does not require further debugging. To do that, we ensure a functional correspondence between a simulated environment and an actual one by requiring both implementations to be in conformance with the same DiaSpec specification.

Generated simulation support. A DiaSpec specification is used to generate both an emulation layer to execute applications and a simulation programming framework to develop simulated entities. Furthermore, simulation scenarios are defined in a *scenario editor* which is parameterized by a DiaSpec specification. Simulated services and stimuli can then be graphically defined using a wizard. To facilitate the creation of simulated services and stimulus producers, we provide libraries of predefined behaviors.

Hybrid simulated environments. DiaSim builds simulated environments as extensions of actual environments. Because of this inheritance strategy, an application can be executed in an hybrid environment, combining simulated and actual services. This feature is particularly useful to perform unit testing of actual entities.

Simulation renderer. We present a simulation renderer that enables the developer to visually monitor and debug a pervasive computing system. The simulation renderer takes into

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

account various features of the pervasive computing domain. Specifically, it supports visual representations for an open-ended set of entities and stimuli, visual support for scenario monitoring, and debugging facilities to navigate in scenarios in terms of time and space.

The DiaSim simulator architecture is shown in Figure 1. Stimulus producers emit stimuli of various types according to a predefined scenario. In place of actual sensors, simulated ones process these stimuli and produce events. The unchanged application reacts to these events by invoking actuator commands. In turn, actuators change the simulated environment, triggering stimulus producers.

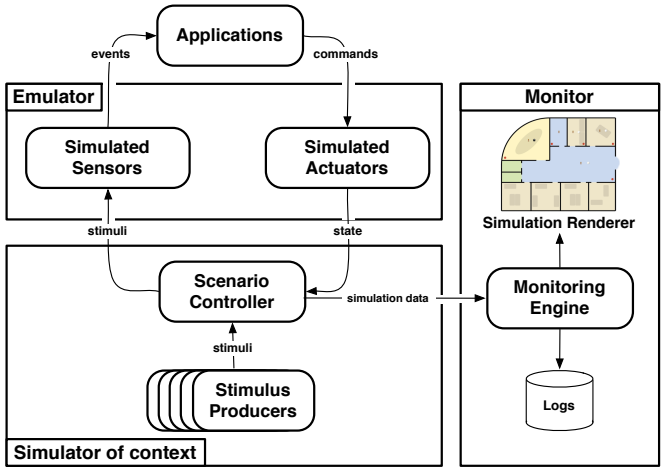

Fig. 1. The DiaSim simulator architecture

III. Demonstration

To demonstrate the key features of DiaSim, we will simulate various applications in the building management area. DiaSim will be used to test these applications and evaluate the feasibility of their deployment in ENSEIRB, an engineering school to which the authors are affiliated.

A. Target environment

The ENSEIRB school is a three-floor building of 13,500 m^2, consisting of several lecture halls, labs and recreation rooms for students. ENSEIRB hosts up to 900 occupants, including students and faculty members. Many devices compose the ENSEIRB school. Sensors such as motion detectors, light and temperature sensors provide context information from all over the building to various applications. The applications trigger several types of actuators. These include alarms, lights, LCD screens, PDAs, loudspeakers and air conditioners.

B. Demonstration Steps

Specifying simulated environments. The first step of our demonstration will show how the DiaSim simulator is parameterized with respect to the DiaSpec specification of the ENSEIRB environment. We will show how to model a pervasive computing environment in DiaSpec and how the resulting DiaSpec specification parameterizes the DiaSim simulator.

Defining simulation scenarios. The second step of our demonstration will be the definition of simulation scenarios. Scenarios will be defined using a Java GUI called the scenario editor (Figure 2). From a DiaSpec specification, simulated services are either graphically defined using a wizard, or developed using the generated simulation programming framework. The demonstration will show examples of both ways to define simulated services. The scenario editor also supports the definition of stimulus producers by allowing the user to define stimulus intensities in areas of the simulated space at specific moments in time. For example, a producer of motion stimuli simulates a user moving in a school hallway at a given time. Furthermore, the definition of simulation scenarios is supported by libraries of generic simulated services and stimulus producers which can be parameterized following the target simulation scenario.

To see how the applications behave in a realistic environment, the simulated scenario defines hundreds of simulated persons moving around in the school. The simulated persons are students and faculty members. Students belong to one of four departments at ENSEIRB. Their behavior depends on the agenda of their study field. When a class they have to attend begins, they go to the class room. Furthermore, they modify the context of the simulated environment. For instance, a simulated person is detected when passing in front of a motion detector. As another example, simulated persons increase the CO_2 density in the room where they are located.

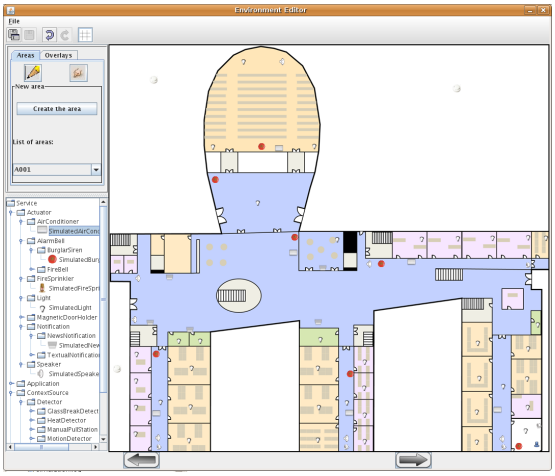

Fig. 2. The scenario editor

Testing applications. Using the programming framework generated from a DiaSpec specification, we developed various applications which will be shown in our demonstration. In the third step, these applications will be tested against the previously defined simulation scenarios. Each application requires several simulated services. To follow the evolution of the simulated environments, DiaSim extends an existing visualization tool: the Siafu open source context simulator [1]. Siafu provides a 2D rendering and time-control functionalities.

[1] http://siafusimulator.sourceforge.net/

Figure 3 shows the resulting simulation renderer. Services and stimuli defined in the scenario are displayed on top of a picture of the simulated space. The simulation renderer gives the state of services by displaying a bubble of raw text above services and/or modifying the visual representation of the service. To complement these macroscopic views, we enriched Siafu's rendering functionalities with Web Interfaces and audio streams. As shown in Figure 3, in the ENSEIRB simulation, clicking on school LCD screens runs a Web interface showing its display. Similarly, loudspeakers are rendered using audio streams.

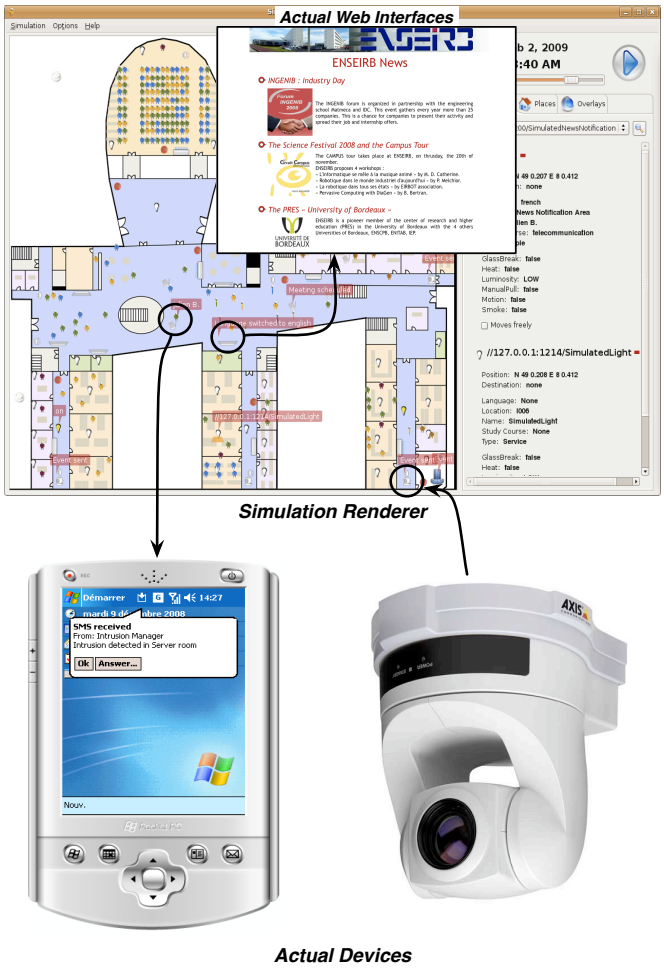

Fig. 3. Demonstration of a hybrid simulated environment

Demonstrating hybrid simulated environments. One of the key feature of DiaSim is to enable hybrid simulated environments, *i.e.,* combining simulated and actual devices in the same simulation scenario. As illustrated in Figure 3, in the final step, we will demonstrate this feature by making actual devices interact with simulated ones. These actual devices are an IP camera and an actual PDA. The camera is used as a motion detector for the intrusion manager. Messages are received on the actual PDA when an intrusion in the simulated environment is detected. These actual devices will be added incrementally in the simulation, demonstrating the flexibility of our approach.

C. Demonstrated Applications

The first application is a newscast manager. Its purpose is to display news and schedules on screens scattered throughout the school. This manager reads the school RSS feed in order to get the latest school news. It also receives schedule events from the ENSEIRB agenda. This agenda describes the courses of the four ENSEIRB departments. The newscast manager alternatively displays the news and the students' schedule. A key feature of the newscast manager is that it adapts the contents displayed on the screens with respect to the department affiliation and the nationality of the people surrounding the screens. Thus, for instance, the messages displayed on a screen are in English if this screen is mainly surrounded by English native speakers. Moreover, when a class begins, the newscast manager plays a message on the loudspeakers of the school using the text-to-speech technology.

The second application is an intrusion manager. It is activated when ENSEIRB is closed. The intrusion manager receives intrusion events from the motion detectors. They are installed in every corridor and room of the school. When a motion is detected, the intrusion manager turns on the school alarms. It also displays a message on the PDA of the school supervisor to warn him and to indicate where the intrusion took place.

The last application is a building automation manager. This manager controls the lights depending on the outside luminosity and the area occupancy. It also controls the school air conditioners to regulate the temperature based on temperature values produced by multiple temperature sensors. When the school is closed, all lights are turned off and the target temperature is lower (in winter) or higher (in summer) than during the day to save energy. Between 7 am and 8 pm, its behavior depends on luminosity values of the outside light sensors. If the luminosity is lower than a predefined threshold, the lights of a given area are turned on, when some people enter this area, and turned off, when the area is empty. If the outside luminosity is greater than the predefined threshold, the previous behavior only occurs for areas without windows. The manager also receives information from CO_2 sensors. When the CO_2 density is too high in a given area, the corresponding air conditioner is activated to decrease its density.

D. Technical constraints

The demonstration does not have special requirements other than a demo booth or table and an Internet connection.

References

[1] W. Jouve, N. Palix, C. Consel, and P. Kadionik. A SIP-based programming framework for advanced telephony applications. In *Proceedings of the 2nd LNCS Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm'08)*, Heidelberg, Germany, July 2008.

[2] W. Jouve, J. Lancia, N. Palix, C. Consel, and J. Lawall. High-level programming support for robust pervasive computing applications. In *Proceedings of the 6th IEEE Conference on Pervasive Computing and Communications (PERCOM'08)*, Hong Kong, China, March 2008. WiP session.